A Collections III

Verwende das Repository `htl3r-####/SEW-2425`

Jeweils ein Unterordner (*package*) je Übung.

A.1 Aufgaben

A.1.1 Cocktailbar

In einer Bar werden unterschiedliche Cocktails gemixt. Für die Cocktails sind verschiedene Zutaten notwendig. Leider sind nicht immer alle Zutaten verfügbar, und der Barkeeper muss ermitteln, bevor er zu arbeiten beginnt, welche Cocktails mit den vorhandenen Zutatene gemixt werden können. Die Daten für die Cocktails können aus dem alten Kassensystem nur über Textdateien exportiert werden.

Um den Barkeeper zu unterstützen, erstelle eine Java Applikation, die die Daten der Cocktails und die vorhandenen Zutaten aus einer Textdatei einliest und dann die Liste der möglichen Cocktails sortiert in der Konsole ausgibt.

Schreibe die Methode

```java
public static Set<String> getAvailableDrinks(String filename)
```

die eine Textdatei einliest und die verfügbaren Cocktails als Set zurück liefert.

Beispieldateien (**CocktailMix.txt** bzw. **CocktailMix_short.txt**) findest du im Anhang.

- **Dateiformat:**

  ```
  Name der Zutat:Namen der Cocktails mit dieser Zutat
  ⋮
  Name der Zutat:Namen der Cocktails mit dieser Zutat
  vorhandene Zutaten
  ⋮
  vorhandene Zutaten
  ```

- **Beispieldatei (CocktailMix_short.txt)**
  (Zeilen die mit »#« beginnen, sind Kommentarzeilen und sollen ignoriert werden, Leerzeilen ebenfalls):

  ```
  # exportiert von Günther Hölzl
  lime juice:Caipirinha,Margarita
  cachaca:Caipirinha
  contreau:Margarita
  tequila:Margarita

  lime juice, cachaca
  ```

  Ausgabe:

  ```
  Caipirinha
  ```

- **Tipps**
  - Verwende eine Map zum Speichern aller Cocktails

Map	Key (~ Index)	Value (= Inhalt)
drinks	Cocktail-Name	String-Liste/Set mit den Zutaten

– Verwende Sets zum Speichern der verfügbaren Zutaten und Cocktails

Set	Inhalt
availableIngredients	verfügbare Zutaten
availableDrinks	verfügbare Cocktails

– es gibt `Set.containsAll(Collection<?> c)`

A.1.2 Klasse: LongCounter

In der Java-API fehlt leider eine Klasse (Map), mit der einfach gezählt werden kann. Schreibe daher die Klasse **LongCounter**, mit der sich beliebige Werte zählen lassen.

- Gehe von der angehängten `LongCounter.java` Datei aus und vervollständige sie.
- Beachte dazu die JavaDoc-Kommentare für diese Klasse.

Anmerkungen

- `LongCounter` erbt von `HashMap`. D.h. du kannst die Methoden der HashMap auch verwenden, bzw. wenn du eine Methode überschreibst mit `super. ...` auf die Methode der HashMap zugreifen.
- `K` ist der "Platzhalter" für den Datentyp des Schlüssels. Diesen Datentyp geben wir beim Erzeugen eines `LongCounter` an. Wo du im Code normalerweise einen konkreten Datentyp (z.B. `String`) verwenden würdest, verwende den Platzhalter `K`.
- Das Fragezeichen (`?`) in der Deklaration `Map<? extends K, ? extends Long> m` wird als **Wildcard** bezeichnet und stellt eine flexible Platzhalter-Syntax in Java generischen Typen dar. In diesem Fall handelt es sich um eine **bounded wildcard**, die den generischen Typ einschränkt.

  – Bedeutung von `Map<? extends K, ? extends Long>`

  * `? extends K`: Das bedeutet, dass der Typ, den die `Map` als Schlüssel verwendet, irgendeine Klasse oder ein Untertyp (d.h. eine Klasse, die von `K` erbt) von `K` sein kann.
  * `? extends Long`: Das bedeutet, dass der Typ, den die `Map` als Werte verwendet, irgendeine Klasse oder ein Untertyp von `Long` sein kann.

  Zusammengefasst erlaubt diese Deklaration der Methode, eine `Map` zu akzeptieren, die Schlüssel von beliebigem Typ verwendet, solange diese vom Typ `K` oder einer Unterklasse von `K` sind, und Werte vom Typ `Long` oder einer Unterklasse von `Long`.

  – Warum wird das verwendet?

  Das Hauptziel dieser `? extends` Syntax ist Flexibilität. Es erlaubt der Methode, nicht nur `Map<K, Long>`, sondern auch `Map<SubTypeOfK, SubTypeOfLong>` als Argument zu akzeptieren. Das bedeutet:

  1. **Mehr Flexibilität beim Typ**: Du kannst mit verschiedenen Typen arbeiten, die mit `K` oder `Long` kompatibel sind, ohne die Methode auf einen ganz bestimmten Typ festzulegen.
  2. **Erweitertes Polymorphismus**: Du kannst Oberklassen und Unterklassen zusammen verarbeiten, was bei der Arbeit mit Vererbungsstrukturen wichtig ist.

  – Beispiel

  Nehmen wir an, `K` sei `Number`. Dann könnte die Methode `putAll()` mit folgenden Maps arbeiten:

```
Map<Integer, Long> integerMap = new HashMap<>();
Map<Double, Long> doubleMap = new HashMap<>();
Map<Float, Long> floatMap = new HashMap<>();
```

  Alle diese Maps sind gültig, weil `Integer`, `Double` und `Float` Unterklassen von `Number` sind.

Zusammengefasst: `? extends K` und `? extends Long` erlauben es der Methode, eine breitere Palette von Typen zu akzeptieren, die eine Beziehung zu `K` oder `Long` haben, was die Wiederverwendbarkeit und Flexibilität des Codes erhöht.